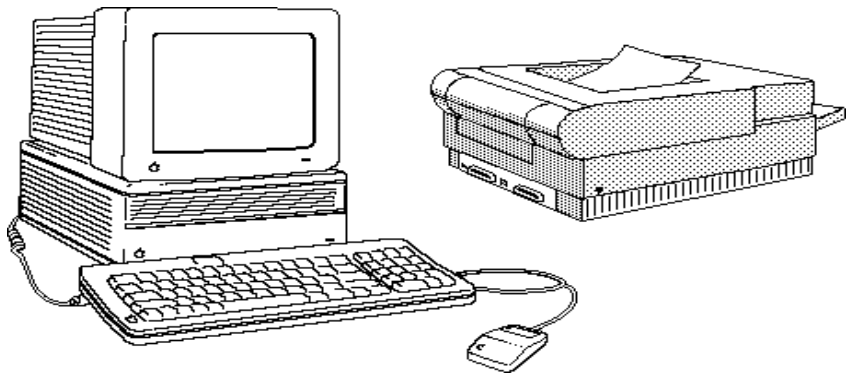
Introduction aux systèmes d'exploitations

Notions de Système d'exploitation

• Un Ordinateur = matériel (**HardWare**) + Programmes (**SoftWare**)



• Les Programmes =

Logiciels de Base + Logiciels d'application

• Les logiciels de Base =

Logiciels de gestion de l'ordinateur simplifiant l'accès au HardWare

• Logiciels d'application =

Nos Programmes + ...

Un système d'exploitation est un logiciel de Base
Il se positionne entre l'utilisateur et le matériel.

Différents types de S.E.

• Mono utilisateur : petite taille, pour ordinateurs individuelles

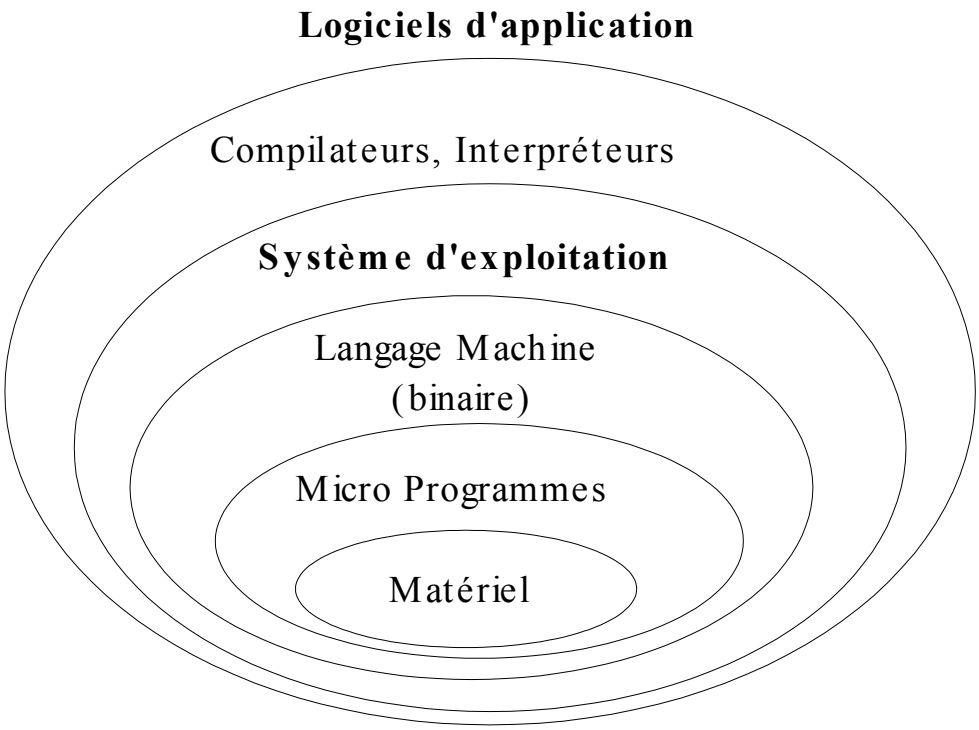
MSDOS, MacOs, OS/2, Win 95, Win NT

• Transactionnel : systèmes bancaires, réservation de billet SNCF, ..

• Multi-utilisateurs : Unix, VMS, MVS (Time sharing)

• Temps Réel : VRTX, OS9000, iRMX, ...

Différentes couches d'un système informatique



Objectifs d'un système d'exploitation

Permettre et faciliter l'accès et l'utilisation de l'ordinateur :

- Utilisation des outils (programmes)
- Création et enrichissement des outils (programmes)
- Gestion des ressources et le matériel (Processeur, Imprimante, Clavier, Souris, Modem, Mémoire, Ecran, Disques, Cartes ....)
- Gestion de la communication et réseaux

Différents types d'utilisation des ordinateurs

- **Exploitation :**

  On ne produit pas de logiciel.

  On se sert des commandes du système d'exploitation pour gérer des fichiers, exécuter un programme (de paie, de dessin, de jeux, d'impression, ...)

- **Administration :**

  Gestion des utilisateurs (création de comptes), sauvegarde des disques, ajout et suppression de périphériques, ...

  Gestion de réseau et des communications

  Création éventuelle de logiciels de base pour les utilisateurs, ...

- **Programmation :**

  On produit des logiciels

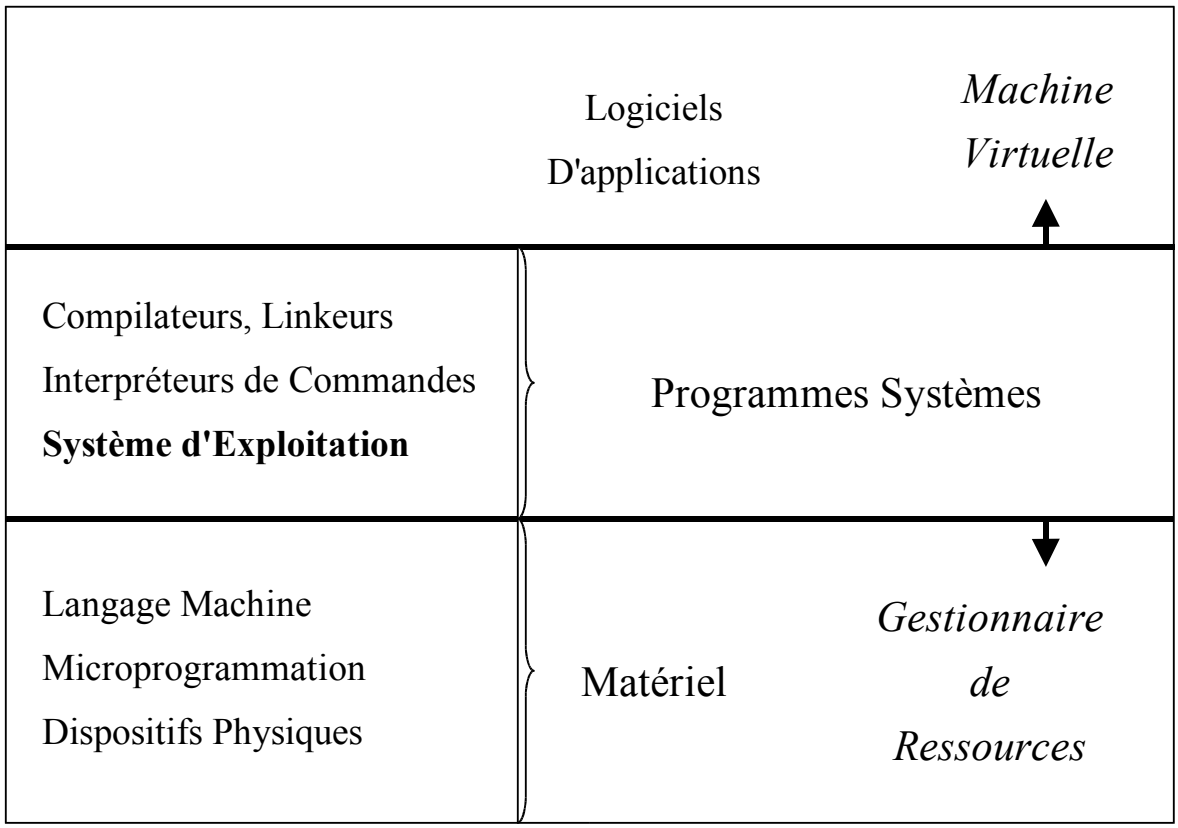
  Les programmes peuvent créer, détruire, modifier des fichiers, lire des données, afficher/ imprimer des données ...

  On se sert d'autres logiciels (éditeur, compilateur, dévermineur) et des commandes du système d'exploitation.

Nous considérons ici la programmation

Le rôle d'un S.E.

Le S.E. libère l'utilisateur de la gestion complexe du matériel.



Un système d'exploitation **Universel** :

- Un noyau petit pour assurer les fonctions de base
- Une large ensemble d'utilitaires
- Une interface utilisateur indépendante (Shell)

Notion de commande

- La communication Homme-Machine se fait par des commandes émises par l'utilisateur vers la machine et par les réponses de la machine.
- Le système d'exploitation prend en charge ces commandes.

Commandes

Utilisateur ==========> Système d'exploitation

- Une commande est une unité **conversationnelle** entre l'utilisateur et le système d'exploitation.

- Une commande est un **ordre**; elle doit être comprise par le système d'exploitation

  Langage et syntaxe des commandes

  Interprétation des commandes

- Le langage de commandes permet d'accomplir différentes activités du système d'exploitation :

  - Développement de programmes
  - Chargement et exécution de programmes
  - Manipulation de fichiers
  - Allocation de périphériques
  - Configuration, protection de l'environnement de travail, des fichiers,...
  - Information, Communication...

Unix : origines

- Création d'un système interactif pour les mini ordinateurs PDP de chez DEC.
- Premier système vers 1970 pour deux utilisateurs sur PDP7.
  => **UNICS** : Uniplexed Information Computing System
- Le langage **C** est crée en 1973 par Denis Ritchie ;
  => Unix est réécrit en C

Caractéristiques d'Unix

• Système d'exploitation **universel** (presque 'indépendant du matériel)
  => un petit noyau gère les E/S et les processus dépendant du matériel

• **Populaire** : bon marché, code **disponible** (pour les universités)

• **Portable** (écrit en C à 90%) : porté sur toutes les machines de toute architecture
  => Existe depuis 1980 pour les PC (XNIX, **Linux**, ...)

• Système **fiable**, mature et riche en outils de développement

• **Interactif** avec une base de logiciels **modulaire**, évolutive
  => amélioration de la productivité (ex : Lex/Yacc)

• Standardisé en 1986 (POSIX : Portable Operating System Interface)

• Interface graphique X-window, Motif, Open-Window, ...

• **Multi-utilisateurs, Multi-Tâches** (multi-traitement)
• Système de fichier **hiérarchique**
• **Protection** des fichiers
• Outils de **communication**
• Langage de commande (**Shell**) puissant

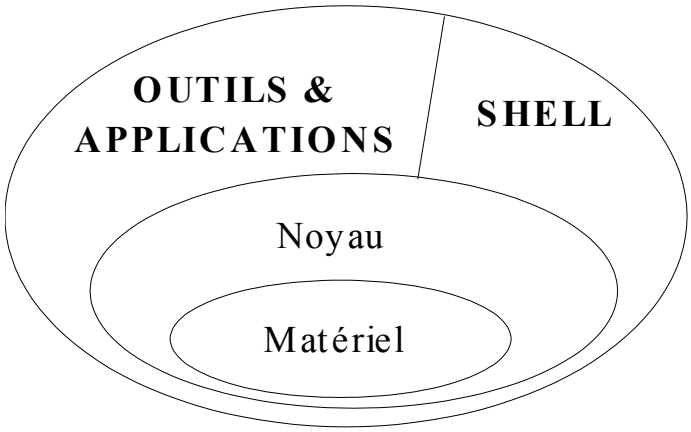
Faiblesses (ou exigences) d'Unix

- Besoin en mémoire et en espace disque (mémoire virtuelle)
- Swapping (baisse des performances)

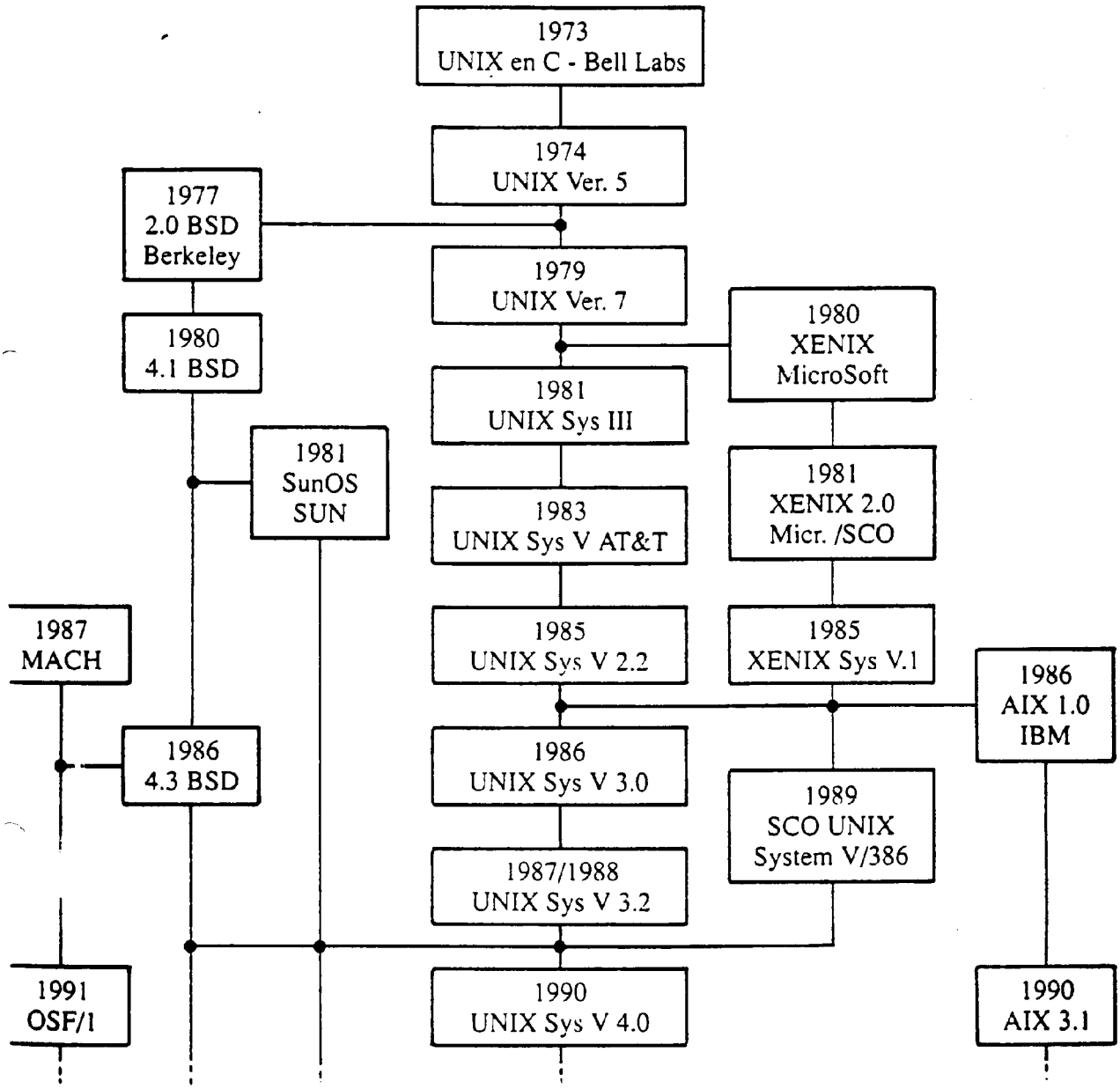
  Evolutions technologique du matériel (prix mémoire/disque)
- Complexité et maîtrise

  Les interfaces conviviales simplifient l'utilisation
- Manque de standard d'interfaces (au-delà de X-window)

Structure d'Unix

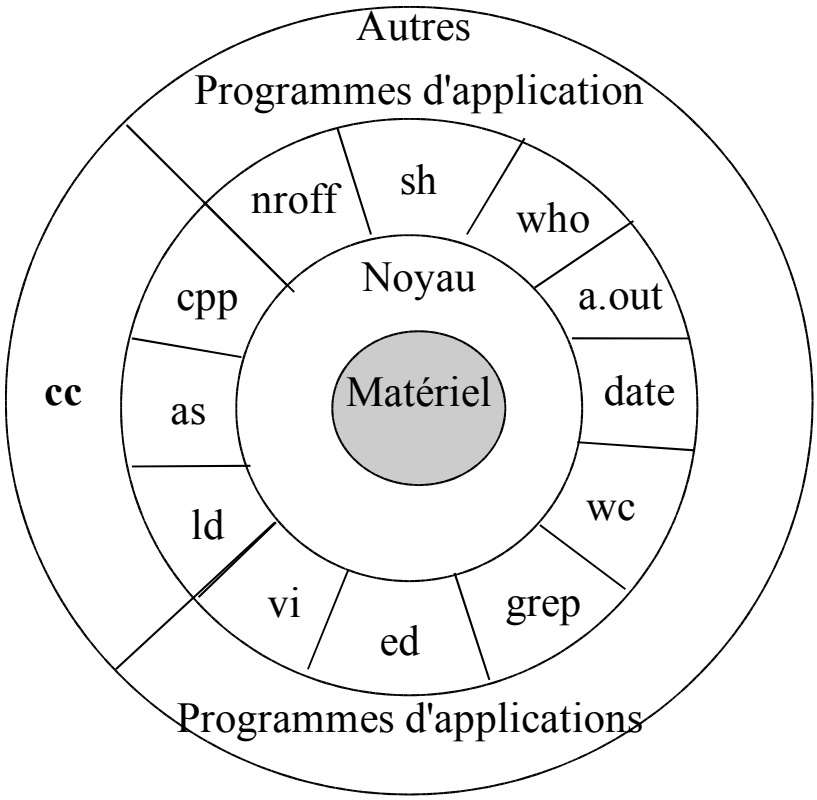


Historique d'UNIX

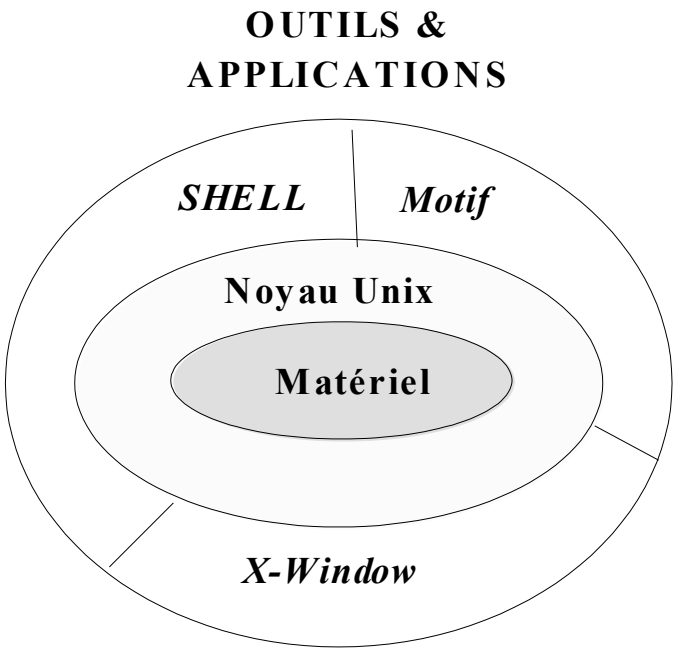


Architecture en couches d'Unix

Quelques exemples d'outils sous Unix.

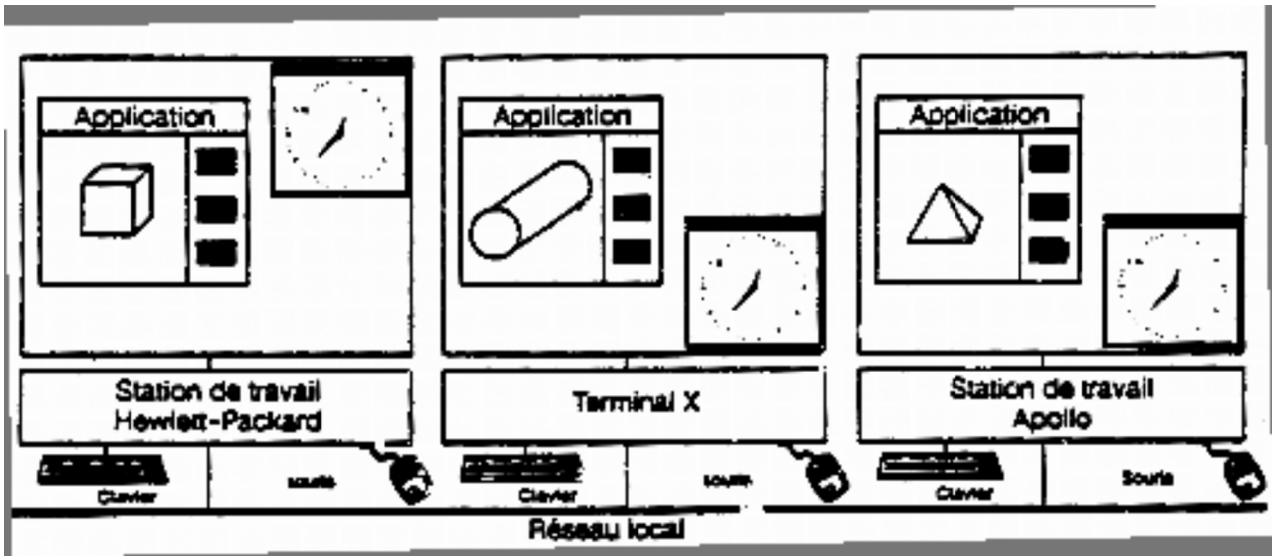


Interface d'utilisateur sous Unix

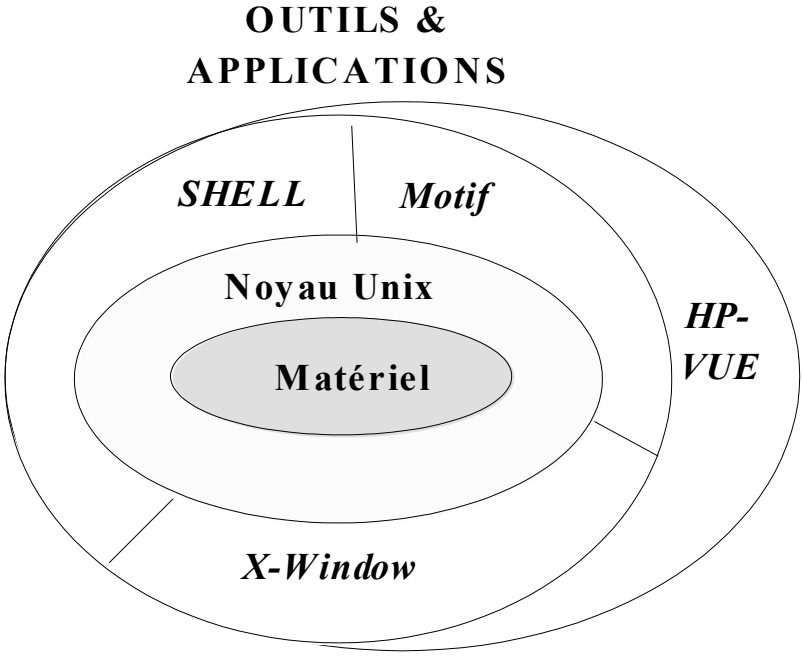


X-Window

- Interface graphique sur un modèle client-serveur orienté réseau.
- Indépendante du matériel et transparente par rapport au réseau.
- Multi-fenêtrage , multi-programmation
- Souple et paramétrable, gestion de la souris / clavier
- Chaque fenêtre est une **machine virtuelle**



Exemple d'Interface sur les HP (HP-UX, HP-Vue)



- Utilisation via l'interface graphique *X-window* ou *HP-Vue*
- Cette interface est un moyen de dialogue convivial entre l'utilisateur et la station de travail.
- Eléments de l'interface :
  - fenêtres, menus déroulants, icônes
  - interaction par la souris et le clavier
  - on entre les commandes par le clavier et la souris.
- HP-VUE permet la manipulation directe des icônes
  par exemple, pour imprimer un fichier, on glisse l'icône du fichier sur celle de l'imprimante.
- On peut se connecter directement sous Shell, sous X-Window ou sous HP-Vue (environnement par défaut).

L'utilisation courante du S.E.

Les opérations classiques offertes :

- Connexion (Login)
- Définition de l'environnement de travail
- Information de l'utilisateur
- Manipulation des fichiers
- Création de programmes
- Exécution de programmes
- Utilisation du réseau (Netscape, FTP, Courrier électronique, ...)
- Déconnexion (Logout)

Connexion - Déconnexion

- Avant toute chose, l'utilisateur d'Unix accède au système en déclinant son identité.

  => il passe par une procédure de "login" semblable à celle exécutée par les distributeurs de billets : carte + code.
- Sous Unix, l'identité = un nom + un mot de passe.

Exemple de login:

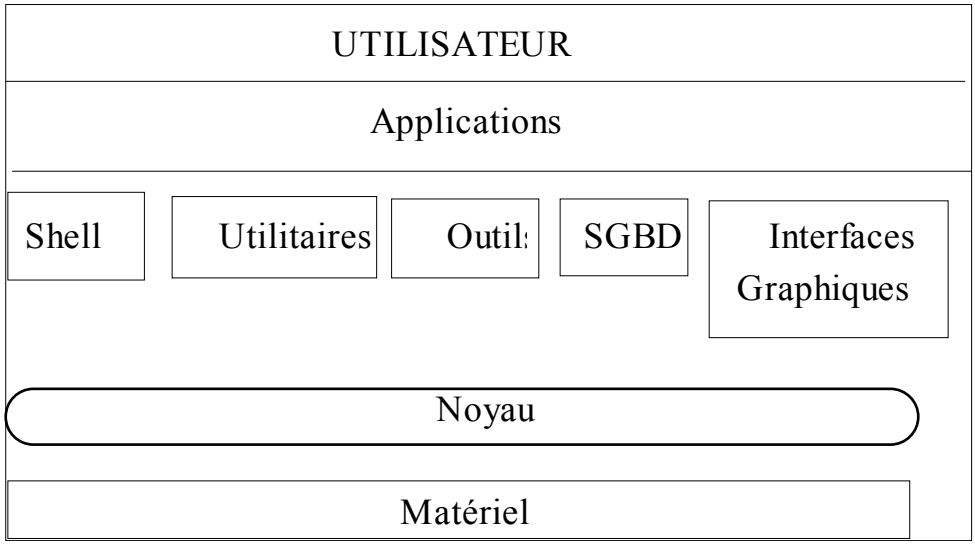
```
Login : durand
Password : #######        (zone illisible)
$                         (le prompt de l'interpréteur)
```

- *durand* doit avoir un compte .
- Si identification réussit, l'environnement de l'utilisateur "durand" est initialisé et le système affiche un "prompt" (invite).
- L'environnement de chaque utilisateur est défini dans des fichiers qui lui sont propres. Il peut les modifier à tout moment.
- Pour quitter le système, on se déconnecte par :

  "logout", **^D** ou "exit".

Eléments du système informatique

Eléments du système informatique tournant sous Unix :



• Shell :

Interpréteur de commandes

• Utilitaires :

Programmes d'exploitation du système (gestionnaire de fichiers, Editeurs, Compilateurs, Outils de communication...)

• Outils :

Programmes d'aide au développement d'applications

• SGBD :

Système de Gestion de Bases de Données

• Interface Graphique :

Ensemble d'outils pour dialoguer avec l'utilisateur

• Applications :

Programmes achetés ou développés.

• Matériel :

Unité système (Unité centrale, Disques, Bandes...), la console système, les Imprimantes, les terminaux Utilisateurs, les lignes de Communication..

• Noyau :

Composant logiciel en contact direct avec le matériel. Toutes les requêtes système passent par le noyau pour exploiter le niveau le plus bas de la machine (matériel). Ceci permet l'indépendance des programmes vis-à-vis du matériel.

Le noyau assure les fonctions suivantes :

- Organisation et gestion des données système
- Organisation de la Protection et l'accès aux données
- Transport d'information entre différents éléments du système
- Supervision du fonctionnement multi-utilisateur et planification de l'exploitation du processeur
- Gestion de la mémoire centrale
- Enregistrement de l'activité du système (mesures statistiques...)

=> Le portage de l'Unix nécessite la réécriture d'une partie du noyau en contact direct avec le matériel.

Quelques concepts de base

Processus

- Unité élémentaire de traitement gérée par le noyau.
- Toute commande (y compris les programmes) est pris en charge par un processus Unix.
- Les programmes peuvent être composés d'un ou plusieurs processus.
- Un processus est composé d'un ensemble d'instructions, des données, d'une pile d'exécution, informations diverses (fichiers ouverts, répertoire courant...)
- Exécution simultanée des processus (Temps partagé)
- Communication et partage entre processus
- Principe d'héritage et de parenté : tout processus est crée par un autre processus.

Fichier (rappel)

- Une collection homogène d'informations regroupées d'une manière logique.
- Les fichiers sont sauvegardée et utilisée dans le système informatique.

Un fichier peut être :

- crées, stockées, supprimées, lues, modifiées, imprimées
- manipulées de manière diverses : protéger, trier, ...

Système de fichiers (File System) d'Unix

• Le File System contient les données et les informations permettant d'assurer la gestion des fichiers.

• Le File System d'Unix est l'entité regroupant les fichiers stockés sur les supports.

• Structure hiérarchique et protégée par des droits d'accès (protections)

• Gestionnaire de l'implantation physique et logique des informations

• Trois type de fichiers (en Unix)

- fichiers ordinaires
- fichiers répertoires
- fichiers spéciaux et périphériques

Organisations physique et logique

structure physique :

organisation sur le support.

structure logique :

la structure de données (dans le programme) qui organise un bloc physique.

En général, l'utilisateur n'est pas concerné par les détails de la structure physique d'un fichier (algorithme d'allocation physique, mécanisme d'accès sur le support, tampons ou buffers, ...)

Commandes

- Cette utilisation est faite par la transmission des commandes aux système via un interpréteur de commandes appelé SHELL.
- L'interpréteur de commande sert d'interface entre le système d'exploitation et l'utilisateur.
- Fonctionnement de l'interpréteur de commandes :

```
Répéter toujours
 - Lire une commande dans une fenêtre F
 - Vérifier la syntaxe de la commande
 - Soumettre la commande au système d'exploitation
 - Transmettre les résultats éventuels à la fenêtre F
```

Classification des commandes

- Commandes générales
- Commandes de manipulation des fichiers et des répertoires
- Commandes de manipulation des périphériques
- Commandes de manipulation de l'environnement et d'état
- Commandes d'administration du système

La structure générale d'une commande

- Une commande est accompagnée des options et des paramètres pouvant être des noms de fichiers :

```
<non-commande> <Options>  <Paramètres>
```

- **Exemple :** *$ wc -l nom_fichier*
  $ ls -al tp-info > resultat
  $
- L'invite ($) indique que l'interpréteur est prêt à recevoir une commande (prompt modifiable, prompt par défaut des Shell)

Quelques commandes par famille

Manipulation des fichiers

cat fic : affiche un fichier à l'écran
mv fic : renomme un fichier
rm fic : détruit un fichier

Manipulation des répertoires

Is : affiche un répertoire à 1'écran
mkdir rep : crée un répertoire
rmdir rep : détruit un répertoire

Environnement utilisateur

logname : nom d'utilisateur
who : qui est logué
date : date et heure
hostname : quel est le serveur

Gestion des processus

ps : process status
kill : destruction de processus

Gestion des entrées-sorties

lpr : lancer une impression

stty : paramètre de la liaison clavier/écran

Administration du système

useradd : ajout user

shutdown : arrêt système

reboot : redémarrage système

mount : montage des volumes (d'un file system)

Les commandes les plus utilisées

ls (ll, lsf) la liste des fichiers
pwd où suis-je ? dans quel répertoire
mkdir création de répertoire
rm suppression de fichier (et répertoire)
cat affichage (re direction)
mv déplacement, re nommage
cp copie
whoami qui suis-je ? (le nom de l'utilisateur)
who qui est connecté au système
cd déplacement dans les répertoires
chmod changement des protections
man manuel des commandes (Exemple **$man man**)
ps état des processus
kill suppression d'un processus
find recherche de fichier
grep recherche dans les fichiers
whereis, **which**
whence recherche d'exécutable / librairie ...
locate, updatedb
recherche d'un fichier sur la machine (à l'aide d'une table mise à jour par *updatedb*)

Commandes générales

date	donne l'heure et la date
cal m a	calendrier du mois **m** de l'année **a**
uname -s	nom du système
uname -a	nom du système, version, état au boot....
who	liste le nom des utilisateurs connectés
whoami	affichage de votre nom d'utilisateur
logname	affichage de votre nom d'utilisateur
id	identificateur d'utilisateur et de groupe (UID, GID)
passwd	permet de changer le mot de passe
tty	nom du terminal (ou du terminal virtuel)
clear	effacement de la fenêtre
echo message	affiche le message
man nom_cmd	manuel du programmeur: (ex : man who)
type nom_cmd	chemin d'accès et type d'une commande
hostname	nom de la station de travail (réseau)

Manuel de l'utilisateur

La commande "man" permet de visualiser la syntaxe et les effets d'une commande.

Exemple : pour demander des renseignement sur la commande "man" :

$man man

man(1) **man(1)**

NAME

man - format and display the on-line manual pages

manpath - determine user's search path for man pages

SYNOPSIS

man [-adfhktwW] [-m system] [-p string] [-C config_file] [-M path] [-P pager] [-Ssection_list] [section] name ...

DESCRIPTION

man formats and displays the on-line manual pages. This version knows about the MANPATH and PAGER environment variables, so you can have your own set(s) of personal man pages and choose whatever program you like to display the formatted pages. If section is specified, man only looks in that section of the manual. ....

OPTIONS

-C config_file

Specify the man.config file to use; the default is /usr/lib/man.config.

Commandes de manipulation de fichiers

cat f1 f2 f3	affiche le contenu des fichiers f1 f2 f3
more f1	affiche page par page le contenu du fichier
pg f1	affiche page par page le contenu du fichier
lpr f1	impression différée(spool d'impression)
cp f1 f2	copie de f1 sur f2; f2 est détruit s'il existait
cp -r f1 f2 r1	copie f1 et f2 dans répertoire r1
mv f1 f3	renomme f1 en f2
rm f1	détruit le fichier f1
rm -i f1	détruit le fichier f1 avec confirmation
rm -r f1	détruit f1 dans tous les sous répertoires
chmod perm fic	changement des droits d'un fichiers
chown fic	changement du propriétaire
chgrp fic	changement de groupe
umask msk	initialisation du masque par défaut
cmp f1 f2	affiche la première différence entre f1 et f2
diff f1 f2	affiche toutes les différences
comm f1 f2	affiche les lignes communes
od f1	affiche le fichier dans son format octal
od -c f1	les octets sont interprétés en car.

Commandes de répertoires

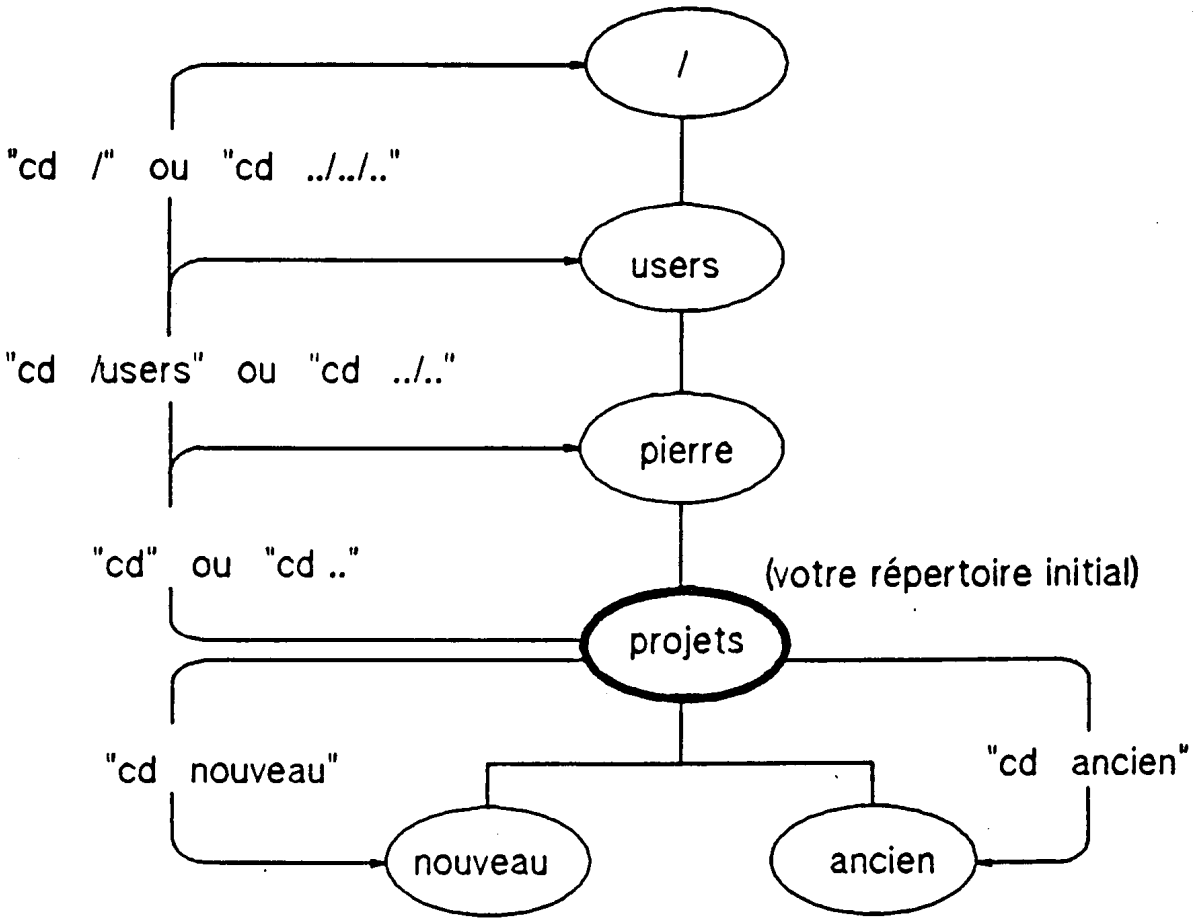
mkdir rep	crée un répertoire rep
rmdir rep	efface le répertoire rep
pwd	affiche le nom du répertoire courant
cd rep	se placer dans le répertoire "rep"
ls	liste les fichiers du répertoire courant
ls fic	liste seulement le fichier nomme
lsf	liste du répertoire (/ pour répertoires)
ls -a	liste aussi les fichiers invisibles
ls -R	liste récursivement tous les répertoires
ls -C	liste en colonnes
ls -x	liste en colonnes triées
ls -t	liste en ordre de mise a jour
ls -u	liste en ordre d'utilisation
ls -i	donne la valeur de l'inode
ls -l ou ll	liste avec toutes les informations (long)

Répertoires UNIX standard

Sont placés sous la racine

bin	programmes exécutables
dev	périphériques
etc	configuration (exemple: /etc/passwd)
usr	répertoire des outils utilisateurs
users	répertoires utilisateurs
tmp	fichiers temporaires (compilations)
lib	bibliothèques

La commande CD



Informations sur les fichiers et les répertoires

file f1	donne le type du fichier f1
du /rep	espace disque utilise par '/rep'
df ou bdf	espace disque libre
ulimit -f	taille maximale d'un fichier
quota –v	l'espace disque autorisé
dirname	partie répertoire d'un chemin d'accès
basename	partie nom de fichier (sans suffixe)

Commandes de processus et signaux

ps processus actifs (process status) du terminal
ps -e tous (every) les processus actifs sur le site (aussi –A)
ps -l processus actifs avec informations complètes
sleep nbsec permet d'endormir le processus nbsec secondes
kill -l liste des signaux
kill -9 pid permet de tuer le processus de numéro pid (process id)
jobs les processus en arrière plan / suspendus (fournit les pids)
kill %job-no suppression du processus N° pid
killall n suppression des processus dont le nom est n

Commande find

Recherche de fichiers

find options paramètres

options:

-name	recherche de fichier par son nom
-print	toujours vraie: le nom trouve sera affiche
-perm	condition sur permission. Exemples
-type	condition sur le type du fichier (b,c,d, p, f).
-links	condition sur le nombre de liens.
-size	condition sur la taille en nombre de blocs de 512 octets
-user	condition sur l'utilisateur
-group	condition sur le groupe

Voir le manuel pour les paramètres (exec, …)

Filtres

head fic	affiche les 10 premières lignes
head -n fic	affiche les n premières lignes de fic.
tail fic	affiche les 10 dernières lignes de fichiers.
tail -n fic	affiche les n dernières lignes de fichiers.
tail +n fic	affiche les lignes a partir de la ligne n.
wc fic	compte les lignes, mots et caractères du fichier fic
wc -l fic	compte les lignes.
uniq	agit sur les lignes répétées d'un fichier
nl fic	numérote les lignes
pr fics	mise en forme des fichiers avec en tête
pr -n fics	mise en forme sur n colonnes.
sort fic	trie les lignes en ordre alphabétique.
grep exp fic	affiche les lignes contenant exp.
grep -v exp fic	affiche les lignes ne contenant pas exp.
grep -n exp fic	affiche les lignes contenant exp avec numéro de ligne
cut -cn-m	sélection des caractères dans les colonnes n a m d'une ligne texte.

Protection et droits d'accès aux fichiers

chaque fichier possède un ensemble de permissions qui déterminent qui peut faire quoi avec un fichier.

Trois types d'utilisateur:

u : propriétaire

g : groupe

o : autres

Trois typés d'utilisation :

r : lecture

w: écriture

x : exécution

La commande 1s -l (ou ll) permet de visualiser les droits d'accès

Exemple des droits sur un fichier (obtenus par ll)

r w x	r - x	r - x	: permissions symboliques
1 1 1	1 0 1	1 0 1	: permission binaires
7	5	5	: même valeur en octal

Commande chmod

chmod permet de modifier les permissions d'un fichier

Exemples

$chmod u-x prog

$chmod g-w fic

$chmod o+r rep

$chmod 700 prog2

Protection répertoires

Trois types d'utilisation

r : liste du répertoire

w : modification (effacement) possible des fichiers

x : droit de traversée (cd)

Définition d'un masque

La commande **umask** permet de positionner un masque définissant les permissions des fichiers lors de leur création.

La valeur du masque est un complément à 1 des permissions désirées. Il s'exprime en base 8

Exemple:

r w x	- - x	- - x	: permissions symboliques
1 1 1	0 0 1	0 0 1	: permission binaires
0 0 0	1 1 0	1 0 0	: masque binaire
0	6	6	: masque octal

Commande umask

La commande umask permet d'installer un masque. Ce masque définit les droits de tous les fichiers crées.

La valeur fournie avec *umask xyz* est le complément par rapport à 7 (pour x, y et z) des droites. Par exemple, 027 définit les droits 750. Pour calculer les droits, on inverse les bits de *xyz*).

Exemple

$umask 027 droits = 750 (complément de 0=7, …)

Compléments

Caractères génériques

***** : remplace toute chaîne dans le nom d'un fichier

? : remplace tout caractère dans le nom d'un fichier

[ccc] : remplace un caractère parmi *ccc*

[c1-c2] : remplace un caractère entre c1 àc2

Exemples

***.c**	désigne tous les fichiers '.c'
prog?.c	les fichiers '.c' dont les noms commencent par prog suivi d'un caractère quelconque
prog.[chC]	les fichier prog.c, prog.h et prog.C
prog[1-9].c	les fichier prog1.c … prog9.c

```
find /users/dupont -name "*.tar" -print
more ch[12345].doc
rm tempo[1-7]
ls tempo?.c
```

Exécution séquentielle

Les commandes peuvent être séparées par un point virgule. Elles seront exécutées l'une après l'autres

```
$date; id; pwd
$cd /usr/include; ls -l | head
```

Capture de la sortie

Une variable peut recevoir le résultat d'une commande mise entre anti apostrophes.

$rep=`pwd` la variable rep contient le résultat de pwd
$echo $rep afficher la variable rep
$list = `ll`
$echo $list
$sys=`uname -a`
$echo $sys

Exécution en arrière plan

Si '&' termine la ligne de commande, celles-ci seront exécutées en arrière plan (on récupère le prompt immédiatement après la frappe).

$cd .. ; du include &

Conjonction et disjonction de commandes (&& et ||)

C1 && C2 : la commande C1 est exécutée, si pas d'erreur, C2 est exécutée sinon stop.
C1 || C2 : la commande C1 est exécutée, si erreur, C2 est exécutée sinon stop.

$cd $HOME /tp3 || cd $HOME/travail && (pwd; cat tp3.sol)

Récupération des erreurs

Redirection par **2> fichier** des erreurs.
$cd ../tp || cd $HOME/travail/tp 2>/dev/null
$ cat f5 || cat f1 2>/dev/null

Les Guillemets

- Les guillemets simples inhibent les commandes

```
$echo '==>echo voila le résultat de la commande ll \n`ll` '
```

==> donnera rien : passe à la ligne et écrit ll (ll = ls –l)

- Les guillemets doubles interprètent les commandes

```
$echo "voila le résultat de la commande ll \n `ll` "
```

==> passage à la ligne et exécute ll

Alias

Un alias permet d'abréger de longues lignes de commandes.
On peut ainsi créer des alias pour les commandes longues et fréquemment utilisées.
Les alias permettent aussi de renommer les commandes pour leur donner des noms plus significatifs.

alias new=anc renomme anc en new
alias liste des alias

Exemple

```
alias lla="ls –la"
alias rename=mv
alias work="cd /users/durand/trav"

alias          donne la liste des 3 définitions ci-dessus
lla = ls –la   …
```

Exercices

Affichage des noms de fichiers par ligne :

ll | cut -c55-

Exercice-1

Cherchez les fichiers dont le nom commence par 'a' ou 'b' et affichez leur taille.

Réponse:

find . -name [a,b] -print*

Exercice-2

Cherchez les fichiers dans le répertoire /usr/local/info dont l'utilisateur est *saidi*. Affichez leur taille et type (répertoires ou exécutables).

Remarque : respecter **{} \;** à la suite de *-exec*

Réponse:

find /usr/local/info -user saidi -exec ls -Fs {} \;

Exercice-3

Même exercice qu'en 2 mais demander confirmation avant l'exécution de la commande.

Réponse:

find /usr/local/info -user saidi -ok ls -Fs {} \;

Exercice-4

Cherchez les fichiers dont la taille en blocs est égale a 5 et affichez leur inode.

Réponse:

find ./ -size 5 -exec ls -i {} \;

Exercice-5

Interpréter les commandes suivantes :

find /example /new/example -exec grep -l 'Where are you' {} \;
find / \(-name a.out -o -name '.o' \) -atime +7 -exec rm {} \;*

Notion de script

- Fichier contenant une séquence de commandes
- Dans le cas simple d'utilisation, un script peut contenir une séquence de commandes sans avoir recours au langage de programmation du Shell.

Exemple : quelques commandes de re direction dans un fichier script

```
#!/bin/ksh          #-- invoquer /bin/ksh pour l'exécution
who  > tempo1
tail < tempo1
head < tempo1 >tempoo
head tempo1 > tempoo
ll /users >tempo2
wc -l < tempo2 >tempo3
echo "L'antislash c correspond au backspace (option –e)"
echo –e "l'exécution de la commande ll sera effectuée à \c"
echo "la suite de ce texte " ; ll
```

Exemple : lister les noms de répertoires du répertoire courant.

```
for i in *              # pour tout fichier
do                      # faire
    if [ -d $i ]        # si c'est un répertoire alors
    then echo $i        # afficher son nom (sinon rien)
    fi                  # fin SI
done                    # fin DO
```

Remarque : le contenu d'un script peut être directement tapé et exécuté au clavier. On utilise un fichier si l'on compte réutiliser cette nouvelle commande.

NB : voir aussi les TPs pour le complément de la syntaxe des Shells.

Utilitaire MAKE

Permet de maintenir, mettre à jour et reconstruire des groupes de programmes.

L'utilitaire MAKE exécute des commandes qui ont été placées dans un fichier make pour mettre à jour un ou plusieurs fichiers cibles.

Les fichiers cibles sont typiquement des noms de programmes exécutables

Exemple de fichier make

```
new_progl : new_progl.C
    CC new_progl.C                tab au début de la ligne
```

Exécution de Make :

$**make** => interprète le fichier par défaut nommé Makefile du répertoire courant

$make -f new_progl.mk

=> Exécute le fichier nommé **new_prog.mk** du répertoire courant

- Les lignes de commandes sont exécutées une par une;
- Un fichier cible n'est mis à jour que s'il dépend de fichiers qui sont plus récents.
- Tous les fichiers dépendants d'une cible sont mis à jour récursivement avant que la cible soit mise à jour. Ceci a pour effet une mise à jour en profondeur de l'arbre des dépendances de la cible.
- L'exécution de make s'arrête dès qu'une commande retourne une erreur (par exemple sur une erreur de compilation dans un programme).

Structure d'un makefile: les règles

Un fichier MAKE peut contenir 4 types de lignes

- lignes cibles
- lignes de commandes Shell
- macro définitions
- lignes d'inclusion

Lignes cibles

Liste de noms de fichiers cibles suivie du caractère ':' et d'une liste de fichiers pré requis ou dépendants.

```
new_progl : new_progl.o bib.o
```

Lignes de commandes Shell

Toutes les lignes qui commencent avec le caractère de tabulation (TAB) sont des lignes de commandes Shell qui doivent être exécutées pour obtenir la cible à mettre à jour.

```
    CC new_progl.o bib.o -o new_prog1
```

L'ensemble de ces deux lignes est un exemple de **règle** :

```
new_progl: new_progl.o bib.o
    CC new_progl.o bib.o -o new_prog1
```

Structure d'un makefile : macro et inclusion

macro définitions

Les lignes de la forme: chaînel = chaîne2 sont des macro définitions. Situées en général en tête de fichier, chaînel est le nom de la macro et chaîne2 est la valeur de la macro

```
OBJETS= filel.o file2.o file3.o
```

Toute occurrence de $(chaînel) est remplacée ensuite par chaîne2

Exemple de fichier makefile avec utilisation d'une macro:

```
OBJETS= filel.o file2.o
new_prog2: new_prog2.C $(OBJETS)
	CC new_prog2.C $(OBJETS)  -o new_prog2

filel.o : filel.C
	CC -c filel.C

file2.o : file2.C
	CC -c  file2.C
```

lignes d'inclusion

Une ligne qui commence par le symbole **include** contient un nom de fichier qui sera traité par le make comme un autre fichier mak après la traduction de toutes les macro éventuellement contenues dans ce fichier.

Un exemple complet

Soit la structure de l'application suivante :

Module Fonc.C

```
#include "fonc.h"
int add( ... )
{//définition
...
}
table tri(...)
{... }
```

Module prog.C

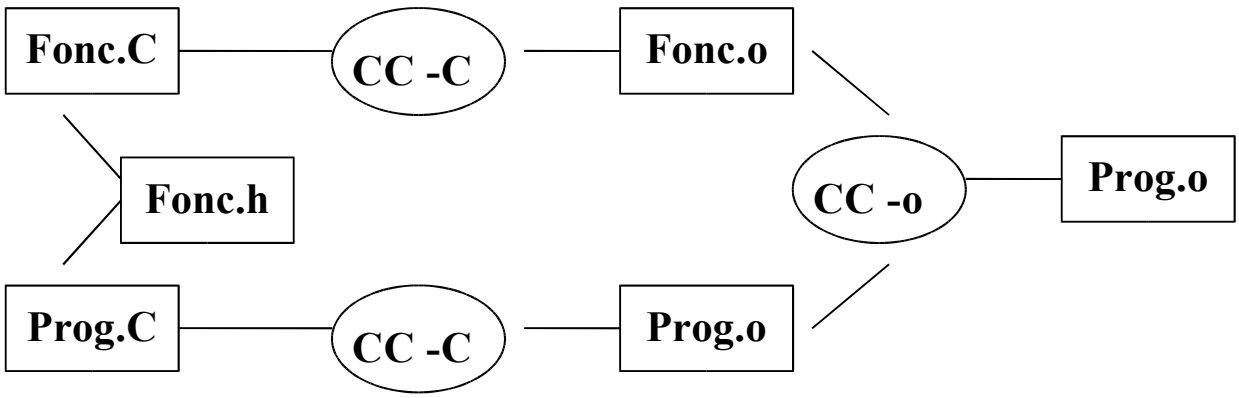
```
#include "fonc.h"
int main()
{ int j; table t;
j= add(....);
t = table(...);
.........
}
```

Fonc.h

```
typedef ... table;
int add( ... );
table tri(...);

//autres définitions
.......
```

Pour construire un exécutable :



Le fichier make correspondant:

```
prog : prog.o fonc.o
	CC -o prog prog.o fonc.o
prog.o : prog.C fonc.h
	CC  -c  prog.C
fonc.o : fonc.C fonc.h
	CC  -c  fonc.C
```

Bibliographie (Partielle)

- ARMSPACH J.P. "UNIX initiation et utilisation"; lnterEditions
- JANSSENS. "UNIX sous tous les angles" . Eyrolles
- BOURNE S. "Le système UNIX" . lnterEditions
- CHAUVIERE J.R. "UNIX présentation et langage de commandes. Dunod.
- CHAUVIERE J.R. " Développer sous UNIX . Outils pour la production de logiciels" EdiTest.
- RIFFLET. "Les communications sous UNIX" . Mc Graw Hill

Documentation

- Doc En Ligne

- Aide par la commande (man)